

PR #37971 完整报告

sgl-project/sglang

fix(glm4v): disambiguate mixed image video offsets

合并时间: 2026-09-05 13:16

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/37971>

执行摘要

- 一句话: 修复 GLM-4V 混合图像 / 视频请求的偏移量分配歧义, 防止运行时崩溃。
- 推荐动作: 此 PR 是一个重要的正确性修复, 解决了会导致服务崩溃的严重 bug。代码变更清晰、集中, 并提供了全面的测试。建议精读 `Glm4vImageProcessor.get_mm_item_offsets` 的实现, 理解其利用边界标记进行模态区分的巧妙方法。同时, 可以关注基类中 `get_mm_item_offsets` 钩子的设计模式, 它为解决类似模型特定差异提供了一个良好的架构扩展点。

功能与动机

GLM-V 系列模型在分词前使用不同的占位符表示图像和视频, 但在分词后, 视频帧使用与图像相同的 token ID (`IM_TOKEN_ID`)。当前的通用多模态偏移量收集逻辑仅根据 token ID 查找占位符范围, 导致在混合图像 / 视频请求中, 图像和视频项都会获取到所有图像 token 的范围, 从而在分块预填充时引发嵌入长度不匹配的崩溃 (Issue #37968)。

实现拆解

1. 在基类引入可覆盖的钩子: 在 `python/sglang/srt/multimodal/processors/base_processor.py` 的 `BaseMultimodalProcessor` 类中, 新增了 `get_mm_item_offsets` 方法。该方法作为分配占位符偏移量到特定模态的入口点, 默认实现基于 token ID 查找, 保持了现有处理器的行为。
2. 修改偏移量分配调用点: 在 `collect_mm_items_from_processor_output` 方法中, 将原本直接调用 `get_mm_items_offset` 并手动查找 token ID 的逻辑, 替换为调用新引入的 `get_mm_item_offsets` 钩子, 从而为子类提供了定制化能力。
3. 在 GLM4V 处理器中实现歧义消解: 在 `python/sglang/srt/multimodal/processors/glm4v.py` 的 `Glm4vImageProcessor` 类中, 重写 `get_mm_item_offsets` 方法。当图像和视频的 token ID 相同时, 该方法会利用 GLM4V 模型特有的视频边界标记 (`VIDEO_START_TOKEN_ID` 和 `VIDEO_END_TOKEN_ID`) 来确定哪些共享的标记跨度属于视频, 从而将偏移量正确归类到 `Modality.VIDEO` 或 `Modality.IMAGE`。
4. 添加全面的单元测试: 新增测试文件 `test/registered/unit/multimodal/test_glm4v_mixed_offsets.py`, 覆盖三种关键场景: 图像后跟视频、交错排列的图像与视频、以及图像和视频 token ID 不同时的回退行为, 确保了修复的正确性和向后兼容性。测试已注册到 CPU CI 套件。

关键文件：

- python/sglang/srt/multimodal/processors/glm4v.py（模块 多模态处理器；类别 source；类型 core-logic；符号 get_mm_item_offsets, is_video_offset）：核心修复逻辑所在，实现了基于视频边界标记的偏移量歧义消解。
- python/sglang/srt/multimodal/processors/base_processor.py（模块 多模态处理器；类别 source；类型 core-logic；符号 get_mm_item_offsets）：引入了可覆盖的偏移量计算钩子，并修改了调用逻辑，是架构层面的改动。
- test/registered/unit/multimodal/test_glm4v_mixed_offsets.py（模块 GLM4V 测试；类别 test；类型 test-coverage；符号 _processor, test_glm4v_partitions_shared_image_and_video_token_offsets, test_glm4v_keeps_interleaved_media_offsets_in_their_modalities, test_glm4v_uses_default_offsets_when_token_ids_are_distinct）：新增的测试文件，覆盖了修复的核心逻辑和边界情况。

关键符号：get_mm_item_offsets, is_video_offset

关键源码片段

python/sglang/srt/multimodal/processors/glm4v.py

核心修复逻辑所在，实现了基于视频边界标记的偏移量歧义消解。

```
def get_mm_item_offsets(
    self,
    input_ids: torch.Tensor,
    mm_tokens: MultimodalSpecialTokens,
    modality: Modality,
) -> List[Tuple[int, int]]:
    """Disambiguate image and video spans sharing ``IM_TOKEN_ID``."""
    # 如果模态不是图像 / 视频，或 token ID 本就不同，则回退到基类默认行为
    if (
        modality not in (Modality.IMAGE, Modality.VIDEO)
        or mm_tokens.image_token_id != mm_tokens.video_token_id
    ):
        return super().get_mm_item_offsets(input_ids, mm_tokens, modality)

    # 获取所有共享 IM_TOKEN_ID 的标记跨度
    all_offsets = self.get_mm_items_offset(input_ids, self.IM_TOKEN_ID)
    # 获取视频边界标记之间的所有标记范围
    video_ranges = self.get_mm_items_offset_by_pair(
        input_ids,
        self.VIDEO_START_TOKEN_ID,
        self.VIDEO_END_TOKEN_ID,
    )

    def is_video_offset(offset: Tuple[int, int]) -> bool:
        """判断一个标记跨度是否完全包含在任何一个视频范围内。"""
        start, end = offset
        return any(
            start >= video_start and end <= video_end
            for video_start, video_end in video_ranges
        )

    return [offset for offset in all_offsets if is_video_offset(offset)]
```

```

        video_start <= start and end <= video_end
        for video_start, video_end in video_ranges
    )

```

根据请求的模式，返回属于该模式的偏移量

```
if modality == Modality.VIDEO:
```

```
    return [offset for offset in all_offsets if is_video_offset(offset)]
```

```
return [offset for offset in all_offsets if not is_video_offset(offset)]
```

python/sglang/srt/multimodal/processors/base_processor.py

引入了可覆盖的偏移量计算钩子，并修改了调用逻辑，是架构层面的改动。

```
def get_mm_item_offsets(
```

```
    self,
```

```
    input_ids: torch.Tensor,
```

```
    mm_tokens: MultimodalSpecialTokens,
```

```
    modality: Modality,
```

```
) -> List[Tuple[int, int]]:
```

```
    """Return placeholder offsets belonging to one modality.
```

```

Processors that reuse one token ID for multiple modalities can override
this method and use surrounding boundary tokens to disambiguate spans.
"""

```

```
mm_token_id = mm_tokens.get_token_id_by_modality(modality)
```

```
if mm_token_id is None:
```

```
    raise ValueError(f"No token id found for modality: {modality}")
```

```
return self.get_mm_items_offset(input_ids, mm_token_id)
```

... 在 collect_mm_items_from_processor_output 中，旧的调用方式被替换：

```
# mm_item.offsets = self.get_mm_item_offsets(
```

```
# input_ids=input_ids,
```

```
# mm_tokens=mm_tokens,
```

```
# modality=mm_item.modality,
```

```
# )
```

评论区精华

BBuf 在 review 中指出了测试文件缺少 SGLang CI 运行器所需的 `if __name__ == "__main__":` 入口点块，否则测试收集阶段会失败。作者 DUXin-ece 随后补充了此代码，解决了该问题。讨论简短但涉及了 CI 兼容性的重要细节。

- 测试文件缺少 CI 入口点 (testing): 作者 DUXin-ece 确认并添加了缺失的入口点代码。

风险与影响

- 风险:

1. 回归风险（低）：变更核心逻辑仅限于 Glm4vImageProcessor 的重写，且通过基类钩子机制隔离。对于图像和视频 token ID 不同的处理器，行为保持不变。新增的测试覆盖了主要场景。

2. 性能风险（低）：额外的边界标记查找仅在请求预处理阶段执行一次，不影响模型的执行、CUDA 内核、注意力或解码性能。
3. 兼容性风险（低）：引入 `get_mm_item_offsets` 钩子是向后兼容的扩展，未修改现有处理器的公共 API。对于其他多模态处理器，如果它们重写了旧的 `get_mm_items_offset` 调用方式（未公开），可能会有问题，但根据代码库分析，这种情况可能性极小。
 - 影响：此修复直接影响使用 GLM-4V/Glm5Next 模型并发送包含图像和视频的混合请求的用户。修复前，这类请求会导致服务器返回 HTTP 500 错误并崩溃。修复后，请求可以被正确处理。对于纯图像或纯视频请求，没有影响。从系统角度看，提高了多模态请求预处理阶段的鲁棒性和正确性。
 - 风险标记：核心路径变更，模型特定逻辑修改

关联脉络

- 暂无明显关联 PR